Fortran 的并行扩展

苏乐 房双德 黄元杰

摘要：Fortran 作为世界上第一个通用的高级程序设计语言，已经经历了 50 多年的发展和多种版本的演变，在科学和工程数值计算领域得到了广泛的应用，是迄今为止高性能计算领域最流行的程序语言之一。受限于其发明年代，最初的 **Fortran** 语言并不支持并行计算。因此，为了增加和增强 **Fortran** 用于并行计算的能力以及保证程序的可移植性，学术界和工业界在不同时代都提出了大量的针对以前 **Fortran** 版本的并行扩展，其中一些已成为了正式或业界的标准而得到了广泛的应用。本文将综述从 **Fortran 77** 以来，被业界广泛支持和接受的 **Fortran** 语言的并行扩展及其特点、实现情况和应用效果。

关键字：Fortran 并行计算 程序设计

1 Fortran 语言概述

1.1 Fortran 语言的诞生

在 1954 年以前，几乎所有的程序都是用机器语言或汇编语言编写的。当时，程序员把生成一个有效程序看作是一项复杂而又富有创造力的艺术活动。他们的精力主要都花在克服当时计算机由于技术限制而产生的各种障碍上，如没有变址寄存器，没有浮点操作等。当时，为程序员提供的“自动编程”系统，主要也是关心如何克服上述不足。这些系统允许有浮点指令、变址寄存器，并改进了输入/输出指令，实际上是一种与实际机器有不同操作码的“人工计算机”。这种人工机比实际机器要容易编程，但是所有这些早期“自动编程”系统使用起来开销过大，通常它们会因此使机器运行速度降低 80%~90%。

其实，在 1954 年前后，花在程序员上的投资已接近计算机本身的价值。而且计算机的使用时间有 1/4 到一半是花在程序的排错上。这样，编程和排错就占据了一台计算机运行投资的 3/4。随着计算机的价格越来越便宜，这种状况也变得更加严重。

正是在上述因素的驱使下，1953 年末，约翰 • 巴克斯（John Backus）[1]向他所在的 IBM 公司建议成立一个小型研究小组，研究开发一种更加有效更加经济的编程方法，以改变当时 IBM 生产的 704 计算机编程效率低下的状况。该建议被当时的 IBM 公司老板卡斯伯特 • 赫德（Cuthbert Hurd）采纳，研究小组随即在巴克斯领导下展开工作。在 1954 年中期产生了一种有相当功能和灵活性的初期编程语言规范，这种语言当时称作 IBM Mathematical FORmula TRANslation System（**FORTRAN**，IBM 数学公式翻译系统）。由于当时计算机主要用于科学计算，该语言的设计目标旨在以充分低的代价把含有丰富数学表达式的程序翻译成充分有效的目标程序。该项目的初衷纯粹是为 IBM 内部研究之用，但在公布了该语言的中期报告后，引起 IBM 客户的极大兴趣。于是，IBM 做出决定，保证每个购买 704 计算机的客户在他们的机器上均能使用 **Fortran** 语言。该版本称为 **Fortran I**。

1.2 Fortran 语言的标准化活动

自从 **Fortran** 语言问世以来，根据需要几经发展，形成了很多版本[2~8]。

经过不断发展，**Fortran I** 形成了很多不同版本，其中最为流行的是 1958 年出现的 **Fortran II**。**Fortran II** 对 **Fortran I** 进行了很多扩充（如引进了子程序），在很多机器上得

以实现。其后出现的 **Fortran III** 未在任何计算机上实现。1962 年出现的 **Fortran IV** 对原来的 **Fortran** 作了一些改变，导致 **Fortran II** 源程序在 **Fortran IV** 编译程序下不能全部直接使用，产生了语言不兼容的问题，形成了 **Fortran II** 和 **Fortran IV** 两种程序设计语言共存的局面。

正因为 **Fortran** 满足了现实的需要，所以它传播得很快，在传播和使用过程中不可避免地产生了多种方言版本。各种 **Fortran** 方言的语义和语法的规定不完全一致，这给用户带来了极大的不便。用户迫切希望有能在各种机型上互换通用的 **Fortran** 语言。因此 **Fortran** 语言的标准化工作变得十分迫切。1962 年 5 月，当时的美国标准化协会（ASA，American Standard Association，后来改名为 ANSI—American National Standards Institute，现名为 NIST—National Institute of Standards and Technology）成立了工作组开展此项工作，1966 年正式公布了两个美国标准文本：标准基本 **Fortran** X3.10-1966（相当于 **Fortran II**）和标准 **Fortran** X3.9-1966（相当于 **Fortran IV**）。

由于 **Fortran** 语言在国际上的广泛使用，1972 年国际标准化组织（ISO，International Standard Organization）公布了 ISO **Fortran** 标准，即《程序设计语言 **Fortran** ISO 1539-1972》，它分为三级，一级 **Fortran** 相当于 **Fortran IV**，二级 **Fortran** 介于 **Fortran II** 和 **Fortran IV** 之间，三级 **Fortran** 相当于 **Fortran II**。

Fortran IV（即 **Fortran 66**）流行了十几年，几乎统治了所有的数值计算领域。许多应用程序和程序库都是用 **Fortran** IV 编写的。但很多编译程序并不向这一标准靠拢，它们往往为实现一些有用的功能而忽略标准；另外，在结构化程序设计方法提出以后，人们开始感到 **Fortran IV** 已不能满足要求。**Fortran IV** 不是结构化的语言，没有直接实现三种基本结构的语句，在程序中往往需要用一些 GOTO 语句以实现特定的算法；而且为了使非标准的 **Fortran** 源程序能够移植，产生了“预处理程序”，通过预处理程序读入非标准的 **Fortran** 源程序，生成标准的 **Fortran** 文本，但这种自动生成的 **Fortran** 程序通常让人难以阅读理解。

美国标准化协会在 1976 年对 ANSI X3.9-1966 **Fortran** 进行了修订，基本上把各厂家行之有效的功能都吸收了进去，此外又增加了不少新的内容，1978 年 4 月美国标准化协会正式公布将它作为美国国家标准，即 ANSI X3.9-1978 **Fortran**，称作 **Fortran 77**。1980 年，**Fortran 77** 被接受为国际标准，即《程序设计语言 **Fortran** ISO 1539-1980》。这种新标准并不是各非标准 **Fortran** 的公共子集，而是自成一体的新语言。我国制订的 **Fortran** 标准，基本采用了国际标准(即 **Fortran 77**)，于 1983 年 5 月公布执行，标准号为 GB3057-82。**Fortran** 77 还不是完全结构化的语言，但由于增加了一些结构化的语句，使 **Fortran 77** 能用于编写结构化程序。此外，还扩充了字符处理功能，使 **Fortran** 不仅可用于数值计算领域．还可以适用于非数值运算领域。

因为 **Fortran 77** 有着明显的局限性，为了引入一些新的功能，适应语言的发展，ANSI 在 80 年代初期开始准备制定 **Fortran** 8x 标准。当初为了与前一标准相对应，设想是 x = 8。由于要将 **Fortran 77** 作为一个子集，同时又要确保程序的高效率，其标准化的工作花了十几年，最终在 1991 年通过了 **Fortran** 90 新标准 ANSI X3.198-1991，相应的国际化标准组织的编号为 ISO/IEC1539:1991。新的 **Fortran** 标准废弃了过时的严格的源程序书写格式，改善了语言的正规性，并提高了程序的安全性，功能有更大的扩充，是一个能适应现代程序设计思想的现代程序设计语言。为了保护对 **Fortran 77** 用户在软件开发上的巨大投资，整个 **Fortran 77** 被作为 **Fortran 90** 的一个严格子集。

随着其他程序设计语言的迅速发展，**Fortran** 不再是唯一广泛使用的程序设计语言。然

而，尽管在一些特殊领域使用其他程序语言更为合适，但在科学和工程技术数值计算领域，**Fortran** 仍具有强大的优势。其强大的生命力在于它能紧跟时代的发展不断更新标准，每次新的文本推出都在功能上有突破性进展。例如，**Fortran 90** 不仅仅是将已有的语言进行标准化，更重要的是吸取了一些其他语言的优点。所以，虽然 **Fortran** 语言历史悠久，但仍在日新月异地发展。

随着巨型计算机(向量机和并行机)的异军突起，出现了新的高性能 **Fortran** 语言(HPF)，它是 **Fortran 90** 的一个扩展子集，主要用于分布式内存计算机上的编程，以减轻用户编写消息传递程序的负担。HPF-1.0 的语言定义是在 1992 年的超级计算国际会议上做出的，正式文本是在 1993 年公布的。其后几年的会议上又对它进行了修改、重定义、注释等工作，于 1997 年发布了 HPF 2.0 语言定义。**Fortran 95** 则包含了许多 HPF 的新功能。在 **Fortran 90** 出现之前，在并行机上运行程序需要结合专门的向量化子程序库，或者是依赖 **Fortran** 编译系统进行自动向量化。而 **Fortran 90** 之后，程序员在编程时可有目的地控制并行化。

在 2004 年后，又陆续公布了支持面向对象编程的 **Fortran 2003** 标准，以及再后面的 **Fortran 2008** 标准等。

2 Fortran 77 以来的并行扩展

Fortran 语言发展过程是随着程序设计方法飞速发展而逐渐演进的过程。进入 **Fortran** 标准和并行计算直接相关的扩展在数量上并不多，以下是 ANSI/ISO **Fortran** 标准发展的一个概述。

2.1 Fortran 77

Fortran 77 相对 **Fortran 66** 在许多方面做了重要改进。**Fortran** 最初是为数值计算设计的，**Fortran 77** 扩充了字符处理功能，使之能应用于非数值计算领域。**Fortran** 77 还增加了块 **IF** 语句、**ELSE** 语句、**END IF** 语句等，使写出的程序趋于结构化，可读性加强。此外，**Fortran 77** 还增强了输入输出的功能和文件处理能力，对 **Fortran 66** 标准中的许多部分做了改进（如允许不同类型变量和数值的混合运算，数组下界可以是负数和零，数组下标表达式可以为任意的整型表达式等）。

具体扩充如下[4][9]：

- **CHARACTER** 数据类型（极大地扩展了字符输入和输出以及对基于字符的数据进行处理的工具）；
- **IMPLICIT** 语句；
- **IF** 语句块，以及可选的 **ELSE** 和 **ELSE IF** 从句（提供改进了的对结构化编程的语言支持）；
- **OPEN**，**CLOSE** 和 **INQUIRE** 语句（以改进读写（I/O）能力）；
- 直接访问文件读写；
- **PARAMETER** 语句（以指定常数）；
- **SAVE** 语句（以保存本地变量）；
- 内部函数的通用名称；
- **DO WHILE** 和 **END DO** 语句；
- **INCLUDE** 语句；
- **IMPLICIT NONE** 变量（用于 **IMPLICIT** 语句）；

- 位处理内部函数（基于类似的包含在工业实时 **Fortran**（ANSI/ISA S61.1 (1976)）中的函数）。

1991 年推出的 IEEE 1003.9 POSIX 标准版为 **Fortran 77** 的编程人员提供了 POSIX 系统上的调用，在文件上定义了超过一百种的功能调用，允许存取 POSIX 兼容的进程控制（process control）、信号处理（signal handling）、文件系统控制（file system control）、设备控制（device control）、过程指定（procedure pointing），以及流输入与输出（stream I/O）。

由于这一版本成功地改变了 **Fortran 77** 开发流程，使得原本过于缓慢重复的编程设计可以顺利地应付计算机领域迅速的变化。

2.2 PCF Parallel Fortran

PCF Parallel Fortran 是由布鲁斯 •利热(Bruce Leasure)等人[10]于 1991 年提交给 **Fortran** 标准化委员会的一个并行扩展，该扩展基于 **Fortran 77**，添加了支持共享内存多处理器体系结构的语句，使得程序员可以指定线程数目，标定私有/共享数据，加锁并控制线程顺序。

计算划分方面，主要增加了 **PARALLEL DO** 和 **PARALLEL SECTION** 两个结构。其中 **PARALLEL DO** 用于标记没有依赖的循环，执行中不保证循环下标的顺序。当需要在进程执行并行段落之中进行同步控制、通信时，提供了加锁、指定 **CRITICAL SECTION** 和全局 **EVENT** 来实现。

值得注意的是，**PCF Parallel Fortran** 并没有要求编译器对死锁进行处理，因此程序员需要自己考虑锁或事件可能引发的死锁情况。**PCF Parallel Fortran** 在早期的机器如 Cray T3E 上得到了实现，但其扩展最终没有进入 **Fortran** 标准。

2.3 Fortran 90

Fortran 90 是 **Fortran 77** 的后续版本，经过长时间的延迟，于 1992 年正式发布。**Fortran 90** 对 **Fortran 77** 的修改集中在程序设计方面[5][11,12]：

- 允许自由格式源代码输入，以及小写的 **Fortran** 关键字；
- 引入模块，将有关联的过程和数据组合在一起，使它们可以被其它程序单元调用，包括允许限制一些模块的特定部分访问；
- 添加 **RECURSIVE**（递归）过程；
- 极大地改善了参数传递机制，允许在编译时检查接口；
- 允许通用过程的用户自定义接口；
- 允许操作符重载；
- 引入派生 / 抽象数据类型；
- 添加了新的数据类型定义语法，以指定数据类型和变量的其它属性；
- 可以在表达式和赋值语句中按整体操作数组（或数组节），由此极大地简化了数学和工程计算的编程。这些特性包括整体、部分和通配的数组赋值（比如用 **WHERE** 语句作选择性赋值），数组常数和表达式，用户定义的数组函数和数组构造；
- 动态内存分配可以通过 **ALLOCATABLE**（可分配）属性和 **ALLOCATE**（分配空间）和 **DEALLOCATE**（释放空间）语句实现；
- 引入 **POINTER**（指针）属性、指针赋值和 **NULLIFY**（指针置空）语句以便于创建和操作动态数据结构；
- 引入 **CASE** 结构以支持多分支选择；
- 引入 **EXIT** 和 **CYCLE** 语句以用于按顺序地“跳出”正常的 **DO** 循环重复；

- 标识符长度扩展至最长 31 个字符；
- 添加行内注释；
- 添加用户可控制的可定义的数字精度；
- 添加新的和增强的内部过程。

Fortran 90 并行计算的底层模型是数据并行，其实现需要通过使用数据数组、提供作用于那些并行的数组之上的操作和内部函数，由编译器针对各个特定的硬件体系结构进行优化。从定义上讲，“数据并行”和所谓的 SIMD[1]（单指令多数据）编程方式之间并不存在太大的区别。从某些角度来看，两个术语几乎意味着相同的事情：程序员写下单独一个操作，比如说“＋”，然后编译器让它针对多块数据以硬件所允许的尽可能并行的方式去执行。

任何一种非 SIMD 的并行计算方式一般都被称为 MIMD[2]（多指令多数据）。一个具有 MIMD 特性的并行编程语言将允许几个不同的子程序（作用于数据的不同部分）被同时调用执行。**Fortran 90** 几乎没有 MIMD 结构。一个 **Fortran 90** 编译器可能在某些机器上通过实现一些 **Fortran 90** 的内部函数(例如 pack 或 unpack)来执行 MIMD 代码，但这对 **Fortran 90** 用户来说是隐藏的。一些 **Fortran 90** 的扩展版本，比如 HPF（**High Performance Fortran**）就显式地实现了 MIMD 特性。

2.4 高性能 Fortran（HPF）

超级计算系统发展到现在已经形成了各不相同的系统结构，如分布存储 MPP[3]、共享存储多处理机系统、向量机、大型机和工作站等。要充分发挥这些不同结构计算机系统的能力，就要求程序中能够提供比传统 **Fortran** 和 **Fortran 90** 程序更多的信息。在此之前，很多研究机构已对此进行了广泛研究，其中以美国莱斯大学（Rice University）的 **Fortran D**[13]语言及奥地利维也纳大学的 **Vienna Fortran**[14]语言影响最大。1991 年底，肯 •肯尼迪（Ken Kennedy）和杰弗里 • 福克斯（Geoffery Fox）建议成立一个非官方的组织，来进一步定义这种语言和进行标准化。于是，一个由工业界和学术界联合组成的机构——高性能 **Fortran** 论坛[15]宣告成立。经过两年多的努力，终于在 1993 年推出了一种能够满足上述要求的新的 **Fortran** 语言标准——高性能 **Fortran**（**High Performance Fortran**，HPF）。

HPF[15~17]的目标是为 **Fortran** 语言（主要是 **Fortran 90**）定义一组语言扩充标准，以实现：

- 支持数据并行程序设计；
- 能在非一致存储访问开销的 SIMD 或 MIMD 计算机上获得最高性能；
- 程序代码便于在不同体系结构的计算机间移植。

2.4.1 本质特征

HPF 并行模型的本质特征是单线程、全局内存空间和松散同步。虽然 HPF 最初以 NUMA[4] 为目标，但是其对于非均一访存机器上数据并行的描述信息亦可指导编译器生成分布式内存机器上的代码，当编译器以分布式内存机器为目标机时，HPF 将被编译成 SPMD[5]程序。

HPF 中的新语法结构 **FORALL** 和固有函数被用于支持数据并行，而数据分布等制导信息用于取得 NUMA 机器上的高性能；外部函数则可用于体系结构底层相关的优化。

[1] Single Instruction stream, Multiple Data stream
[2] Multiple Instruction stream, Multiple Data stream
[3] Massive Parallel Processing，大规模并行处理
[4] Non Uniform Memory Access，非均一访存
[5] Single Program, Multiple Data

2.4.2 HPF 的数据分布制导

HPF 有一套编译制导来协助程序员表述数据是如何存储的。这些制导不会影响计算结果，但是会影响到执行效率。它们的引入是基于以下两个重要但是简单的观察：

- 如果同一个操作要处理多个数据对象，那么这些数据对象在同一个处理器上执行效率会更高；
- 如果操作在不同的处理器上同时执行，那么整体的执行效率会更高。

使用这些数据存储的编译制导，我们可以把数据合理地分布在多个处理器上，而编译器则会进行数据并行的优化，从而在运行时的帮助下提高程序的执行效率。

在 HPF 中的主要操作对象是数组，编译制导可以把数据分成相同大小的块，称之为 **BLOCK** 分布，或者按照轮询（round-robin）的方式和其它数组对齐，称之为 **CYCLIC** 分布（或者使用更为复杂的方式进行数组数据的划分）。然后这些数据对象（数组）将会被对齐到一些抽象模板上，这些抽象模板则会被分布在抽象节点处理器上，编译器会把这些抽象处理器映射到物理处理器上，从而完成整个数据的映射过程。如图 1 所示：

图 2 中第一行 HPF 制导信息展示了指示编译器将对于任意 K，将 A(K)和 B(K-1)映射到同一个虚拟处理器上的制导信息。而第二行 HPF 制导信息则要求将数组 C(I, 1:100)复制到存有 D(I, 1:100)中任意元素的虚拟处理器；数据复制操作和数据同一性保证均由编译器保证。

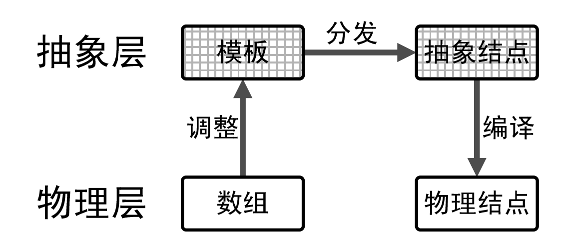


图1. HPF 的数据映射模型示意图

```
REAL  A(100), B(0:101), C(100,100), D(100,100)
!HPF$ ALIGN A(K) WITH B(K-1)
!HPF$ ALIGN C(I,J) WITH D(I,*)
```

图2. HPF FORALL 结构的示例

图 3 中对分块和循环散列两种不同的 **DISTRIBUTE** 方法做了演示，如果目标平台为四处理器的机器，则在第一个处理器上将保存数组的以下部分：`A(1:25,1:100)`、`B(1:100,1:97:4)`和`C(1:5)`、`C(21:25)`、`C(41:45)`、`C(61:65)`、`C(81:85)`。

```
REAL A(100,100),B(100,100),C(100)
!HPF$ DISTRIBUTE  A(BLOCK,*),B(*,CYCLIC),C(BLOCK(5))
```

图3. HPF 通过 DISTRIBUTE 制导信息控制数据分布

2.4.3 HPF 的数据并行结构

HPF 主要并行结构为 **FORALL** 结构，用于计算数组操作，而 **WHERE** 语句可用来控制循环体作用的数组范围。

相比 **Fortran 90** 标准中的数组操作，**FORALL** 语句或结构允许更加灵活的循环下标选

择，如能方便地支持对于二维数组对角线上的元素进行赋值。但是需要指出的是 **FORALL** 结构从设计思路上讲，并不是一个通用的并行结构，它不能很好地支持数据流水计算或者 MIMD 计算。

```
FORALL (I=2:N)
  A(I) = A(I-1) + A(I) + A(I+1)
  B(I) = B(I-2) + B(I) + B(I+2)
END FORALL
```

图4. HPF FORALL 结构的示例

另一点需要注意的是 **FORALL** 结构中的数据依赖。一个有多条子句的 FORALL 结构等价于多个 **FORALL** 语句的序列；而每个 **FORALL** 语句执行时，是在计算对于所有下标的赋值表达式的右值之后，才进行对左值的赋值的。如图 4 中的 **FORALL** 结构的计算顺序为先对任意 I 求 A(I-1) + A(I) + A(I+1)，再将值赋给 A 中的相应的元素，再对任意的 I 求 B(I-2) + B(I) + B(I+2)，最后对 B 中的相应元素赋值。

在 HPF 中由 **INDEPENDENT** 制导标记不依赖可并行的结构，可用于 **Fortran** 的 **DO** 循环或者 HPF 的 **FORALL** 结构。为了保证确定性的结果，**INDEPENDENT** 标记的 **DO** 循环中不可含有对于标量的赋值，除非用 NEW 标记为不跨循环下标使用的临时变量，如图 5 所示。

```
!HPF$ INDEPENDENT, NEW(TEMP)
DO I=1,100
    TEMP=COS(A(I)})
    D(I)=TEMP**2+B(I)
ENDDO
```

图5. HPF 通过 INDEPENDENT 指定并行循环、NEW 指定临时变量的示例

2.4.4 HPF 的其他特性

HPF 还提供了一些内置函数供程序员使用，例如 MAXLOC、MINLOC 用来求最大最小值的索引等。

在语言特征修改方面，**Fortran 77** 的有序存储和现代体系结构中的局部性要求产生了冲突，因此 HPF 取消了对于序列存储的要求，除非通过制导信息制定，编译器可对存储分布进行修改；同时编译器可以利用制导信息优化数据的组织分布。

HPF 是一种不依赖于特定机器的高级语言，为了有效地支持如脉动通信等细粒度优化，还引入了外部函数这样一个接口。在 HPF 2.0 中还增加了 HPF I/O 扩展，但是厂商仍可以定义自己的读写（I/O）扩展。

2.4.5 HPF 的实现情况

随着 HPF1.0 到 1.1，2.0 的提出，广泛应用支持 HPF 的编译器越来越多，而更多的研究机构和厂商也纷纷投入其中，以下列举一些支持 HPF 的编译器：

- Absoft 公司的 Absoft Pro Fortran；
- Digital 公司的 DIGITAL Fortran（不再提供使用）；
- Parsec 技术公司的 **Fortran 90**（不再提供使用）；
- 康柏（Compaq）公司的 Fortran Power Station 4.0（不再提供使用）；
- NA Software Ltd 公司的 HPF Plus；
- 中国科学院的 limaoshan（不再提供使用)）；
- Portland Group 公司的 PGHPF；
- Crescent Bay Software 公司（以前的 Pacific-Sierra 研究院）的 VAST-HPF；
- Applied Parallel 研究院的 xHPF（不再提供使用）；
- IBM 公司的 xlhpf；
- GMD-SCAI 公司的 Adaptor；
- **Fortran 95** 编译器 G95；
- 南安普顿大学和 VCPC（Vienna 并行计算欧洲中心）的 SHPF 等。

很多重要的应用也从 HPF 中获得收益，主要集中在物理、数学等计算密集、数据并行度较大的应用，如：

- 三维磁流体动力学仿真；
- 生物膜仿真；
- 加速器的计算物理学仿真（Computational Accelerator Physics）；
- 莱斯大学的 HPFt 项目-泛型耐撞性仿真内核（Generic Crash Kernels）；
- 癌变过程仿真模型的马尔科夫蒙特卡洛方法（MCMC for Carcinogenesis Models）；
- 预测石油净储量的神经网络（Neural Networks for Hydrocarbon Net Pay Prediction）；
- 普林斯顿海洋模型（Princeton Ocean Model）等。

2.4.6 HPF 的应用效果

戴尔·夏尔斯（Dale Shries）等人[18]用非结构化有限元模拟器（Unstructured Finite Element Simulations）应用把 HPF 和 MPI 的效率进行了对比。实验使用 Cray T3E 超级计算机，编译器采用 PGHPF 3.2。T3E 是一个平行分布式内存的平台，采用了紧耦合的 3D 双向环的配置，拥有 1088 个处理单元，每个处理单元拥有 500MB 内存。实验结果如图 6 所示。

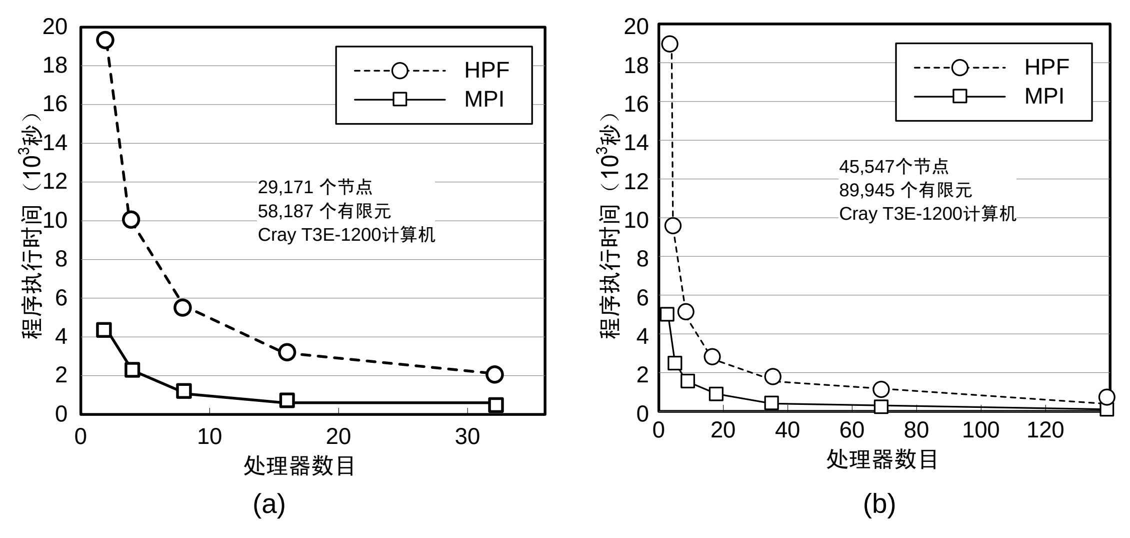


图6. HPF 和 MPI 的执行时间对比图

图 6(a)中，在有 2 个处理器的情况下，MPI 程序比 HPF 程序快 4.5 倍；在有 32 个处理器的情况下，快 4.3 倍。在图 6(b)中，则在有 2 个处理器的情况下，MPI 程序比 HPF 快 3.8 倍；在有 128 个处理器的情况下，快 2.7 倍。

实验结论是，HPF 和 MPI 对于非结构化有限元这类应用都具有较好的扩展性，且这两种并行编程方式都比较适用于大多数并行系统例如 SUN，SGI，IBM 等。而 HPF 是一个较高层次的并行编程抽象，对于 HPF 来说获得这样的执行效率已经是非常不容易的。虽然比起 MPI 应用的执行效率仍有差距，但是 HPF 的编程比 MPI 大大简化，这也大大降低了并行程序设计的门槛，对并行系统的推广起到了很大作用。

村井仁（Hitoshi Murai）等人[19]，在 8 处理器的机器上，使用 HPF/SX V2 编译器，在 HPFBench[20], APR Benchmarks[21], GENESIS Benchmarks[22]和 NAS Parallel Benchmarks[23]上评估了 HPF 的应用效果。图 7 所示为在 HPFBench 上的结果，基准是 F90 不加并行扩展的程序。

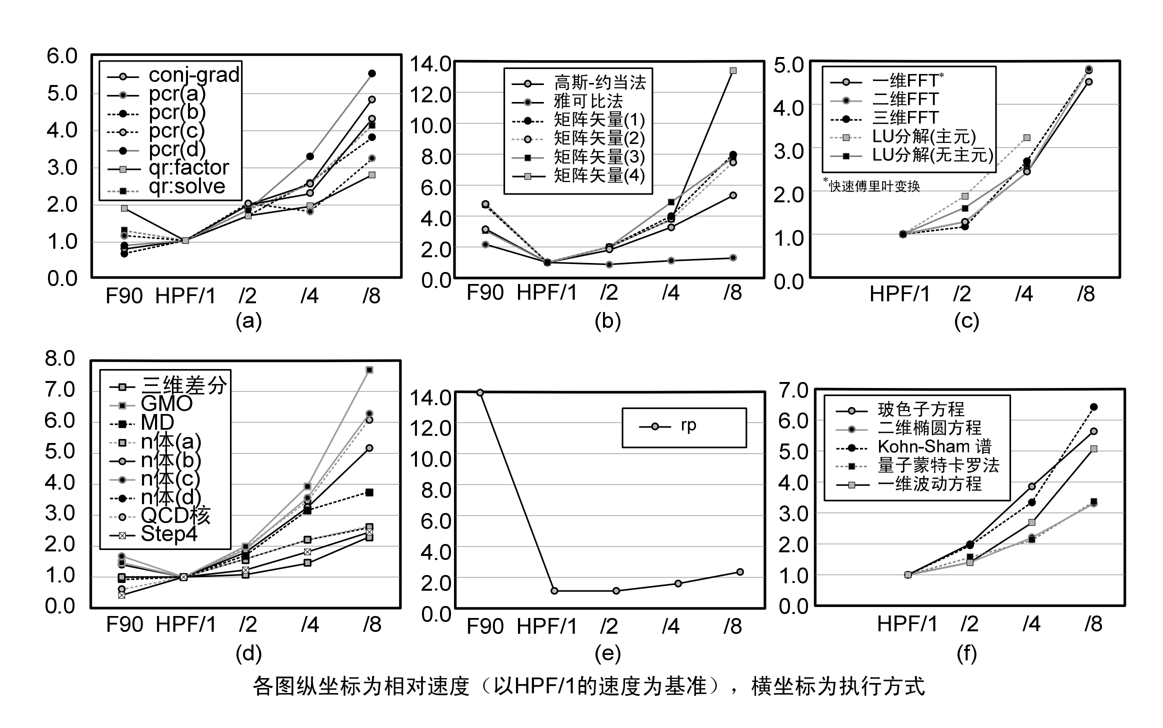


图7. 在 HPFBench 上评估 HPF 的结果[19]
图中(a)~(c)是线性代数库的评估结果；(d)~(f)是应用核的评估结果。

从图 7 的(a)~(f)可以看到总体结果为：在 1 个处理器环境下使用 HPF 扩展时，性能略微有一些下降或者保持不变，在多个处理器环境下使用 HPF 扩展时，可以得到较好的执行效率和扩展性。

2.5 Fortran 95

Fortran 95[6][12][24]仅是一个小改版，大部分改动是修正了 **Fortran 90** 标准中一些较为显著的问题。虽然如此，**Fortran 95** 仍有不少的扩充，尤其是在 HPF 的规格方面：

- 添加了 **FOR ALL** 和嵌套的 **WHERE** 结构以帮助向量化；
- 引入用户定义的 **PURE** 和 **ELEMENTAL** 过程。

Fortran 95 的一个重要补充是 ISO 技术报告 TR-15581：增强的数据类型工具，非正式

名称是可分配的 TR。这一标准定义了 **ALLOCATABLE**（可分配）数组的增强的应用，其出现早于完全兼容 **Fortran 2003** 的 **Fortran** 编译器，使用户可将 **ALLOCATABLE** 数组用于过程伪参数列表及函数返回值中作为派生的类型组件。（**ALLOCATABLE** 数组比基于 **POINTER**（指针）的数组更受欢迎，因为 **ALLOCATABLE** 数组是由 **Fortran 95** 保证的，当它们退出范围时会被自动释放掉，避免了内存溢出的可能性。另外，别名也不再是优化数组引用时的一个问题，可以使编译器生成的代码比用指针时生成的代码运行得更快。）

Fortran 95 的第二个补充是 ISO 技术报告 TR-15580：浮点异常处理，非正式名称是 IEEE TR。这一标准定义了对 IEEE 浮点算术和浮点异常处理的支持。

2.6 Fortran 2003

Fortran 2003[7][24~26]是 **Fortran** 发展中的一次巨大改变，引入了当时流行的面向对象的编程方法。主要的改进体现在：

- 增强了衍生类型：带参数的衍生类型，改善了控制的可操作性，改善了结构化的创建和释放；
- 支持面向对象编程：扩展类型和继承、多态、动态类型分配，以及类型绑定过程；
- 改善了数据操作：支持可分配的组件（编入 IEEE TR 15581），延期的类型参数，**VOLATILE**（易变）属性（在并发系统中，程序员通过添加该属性，可以知道共享数据的取值是否已被正确刷新），支持在数组构造和分配语句中显式定义类型，支持增强的指针、扩展的初始化表达式、增强的内部过程；
- 增强的输入／输出：支持异步传输、流访问，允许用户指定衍生类型的传输操作，用户在格式转换时可以指定舍入控制，为连接前单元指定常数，使用 **FLUSH** 语句，定义了关键字的规范和访问错误信息；
- 支持过程指针；
- 支持 IEEE 浮点算法和浮点异常处理（编入 IEEE TR 15580）；
- 支持与 C 语言的交互性；
- 支持国际化：访问 ISO 106464 字节字符和在格式化的数字输入／输出中选择数字或者逗号；
- 提供与宿主操作系统增强的集成：可以访问命令行参数、环境变量和处理器错误信息。

2.7 Fortran 2008

Fortran 2003 之后的版本是 **Fortran 2008**[8][25,26]，与 **Fortran 95** 一样，只是一个小改版，略微更正了 **Fortran 2003** 的一些问题，并且合并了 TR-19767 的语言功能：

- **Co-array Fortran** – 并行处理模式；
- BIT 数据类型。

2007 年 8 月，数据类型 BIT 被删除了。2008 年 2 月，**Co-array Fortran** 的计划已缩小，仅有 Parallel I/O，而研发团队也被裁员了。

2.8 Co-array Fortran

Co-Array Fortran（简称“CAF”）[27~32]是一组对 **Fortran 95** 的 SPMD 的并行扩展。CAF 的主要并行对象也是数组，且仅仅对原来的 **Fortran 95** 语法进行了很简单的扩展，所以程序员学习的负担很少。

使用 CAF 编写的 **Fortran** 程序段就好像是被复制了多份，每份都异步地执行。每份程序都有自己的数据对象，称为 image（镜像）。CAF 主要是扩展了数组的索引下表，CAF 使用方括号进行跨 image 的访问。

CAF 是一个 SPMD 的并行编程模型，基于 **Fortran 95** 语言。和 MPI 类似的是 CAF 程序需要显式地管理数据和计算的分布，但是 CAF 是一个共享内存的编程模型，且不需要显式地进行数据通信管理，只需要使用对 **Fortran 95** 数组下标的扩展进行其它处理器数据的访问，数据的通信和同步则是交给编译器进行处理和优化。CAF 现在主要在科学、工程计算中应用，在超级计算机中应用较多，主要在 Cray **Fortran 90** 编译器 3.1 版本之后实现。此外莱斯大学的 Los Alamos Computer Science Institute (LACSI) 实验室[28]也实现了 CAF。

2.8.1 Co-array Fortran 的主要特点

首先是 CAF 的计算分布(work distribution)，一个程序被复制多份，每份有自己的一组数据对象，每一份都成为镜像（image），且多个镜像之间是异步执行的，所以每个镜像的执行路径都有所不同。程序员使用镜像索引（image index）和显式的同步操作来决定某个镜像实际的路径。编译器负责对 CAF 进行优化。

其次，考虑数据分布，CAF 扩展了 **Fortran** 语言的语法，允许程序员使用数据下标式的语法来在多个处理器上分布以及存取数据。例如：

```
REAL, DIMENSION(N)[*] :: X,Y
X(:) = Y(:)[Q]
```

上面的代码中，声明的每个镜像有 2 个实数数组 X 和 Y，每个数组大小都是 N，且如果 Q 在每一个镜像上都是相同的值，那么第二句话的意思是每个镜像上的 X 数组都拷贝镜像 Q 上 Y 数组的值。在小括号里的数组下标表示的是在镜像内部，数组元素的索引。在方括号里的数组下标则表示的是数组所在的镜像的索引。使用数组下标的扩展语法使程序员更容易编程存取其他镜像上的数据。

由于一个程序中，许多对数据对象的操作在本地进行才是最高效的，CAF 的语法应该在一个很小的比较单独的部分出现。否则，使用 CAF 的代码越多将标志镜像之间需要的通信越多。这里给出一个 CAF 的简单的例子：

```
1.  X = Y[PE]          ! 从 Y[PE]取数
2.  Y[PE] = X          ! 向 Y[PE]赋值
3.  Y[:] = X           ! 广播 X
4.  Y[LIST] = X        ! 广播 X 覆盖 PE'在数组 LIST 中的子集
5.  Z(:) = Y[:]        ! 收集全部 Y 值
6.  S = MINVAL(Y[:])   ! 求所有 Y 的最小值
```

如上例中，第 1 行把 X 的值赋成第 PE 个镜像中 Y 的值；第 2 行为把 X 的值赋给第 PE 个镜像中 Y；第 3 行将当前镜像中 X 的值赋给所有镜像中的 Y；第 4 行把当前镜像中 X 的值赋给 LIST 所指定的那一组镜像中的 Y；第 5 行把所有镜像中 Y 的值赋给当前镜像中 Z 的值；第 6 行把所有镜像中 Y 的值取得，并求最小值，赋给当前镜像中 S。

在此之前，输入输出对于 SPMD 的编程模型来说是一个比较麻烦的问题，例如 MPI。因为标准的 **Fortran** 输入输出是假定有一个专门的进程来读取文件，这个限制往往会被违背，尤其是当每个镜像的输入输出都要独立进行的时候。而 CAF 使用最小的 **Fortran 95** I/O 的扩展避免了之前程序模型所面临的一些限制，从而可以显式地提供并行读写，且可以基于

进程和线程实现。

图 8 是一个求最大值的 CAF 程序完整实例：

```
Subroutine greatest(a,great) ! 找出 a(:) [*]的最大值
   real, intent(in)::a(:)[*]
   real, intent(out)::great[*]
   real :: work (num_images()) ! 局部工作数组
great = maxval (a(:))
   call sync_all ! 等待所有镜像到达
   if(this_image(great)==1)then
      work(:)=great[:] ! 获得局部最大值
      great[:]=maxval(work) ! 广播全局最大值
   end if
   call sync_all
End subroutine greatest
```

图8. 使用 CAF 扩展求最大值

```
Subroutine greatest (first,last,a,great)!求 a(:)[first:last]的最大值
   integer, intent(in)::first,last
   real, intent(in)::a (:)[*]
   real, intent(out)::great(:)[*] !将结果放入 great [first:last]
   real, allocatable::work(:) ! 局部工作数组
   integer::i,this
   this=this_image(great)
   if (this.GE. fitst.and. This .LE.last) then
      allocate (work(first:last))
      great=maxval(a)
      call sync_team((/(i,i=first,last)/))
      if(this. EQ.first)then
         work=great[first:last] ! 获得局部最大值
         great[first:last]=maxval(work) ! 广播全局最大值
         deallocate (work)
      end if
      call sync_team((/(i,i=first,last)/))
   end if
end subroutine greatest
```

图9. 使用 CAF 扩展求最大值，使用 allocatable array

如图 8 中的代码所示：一开始，所有镜像求得本镜像内部的数组 a 的局部最大值。然

后同步，等待所有的镜像都完成最大值的求解，然后第一个镜像去收集所有镜像中的局部最大值，并且求得全局最大值，然后分发给所有镜像。第 6 行中的 **call** `sync_all` 是 CAF 提供的方法，用来进行镜像间的同步。

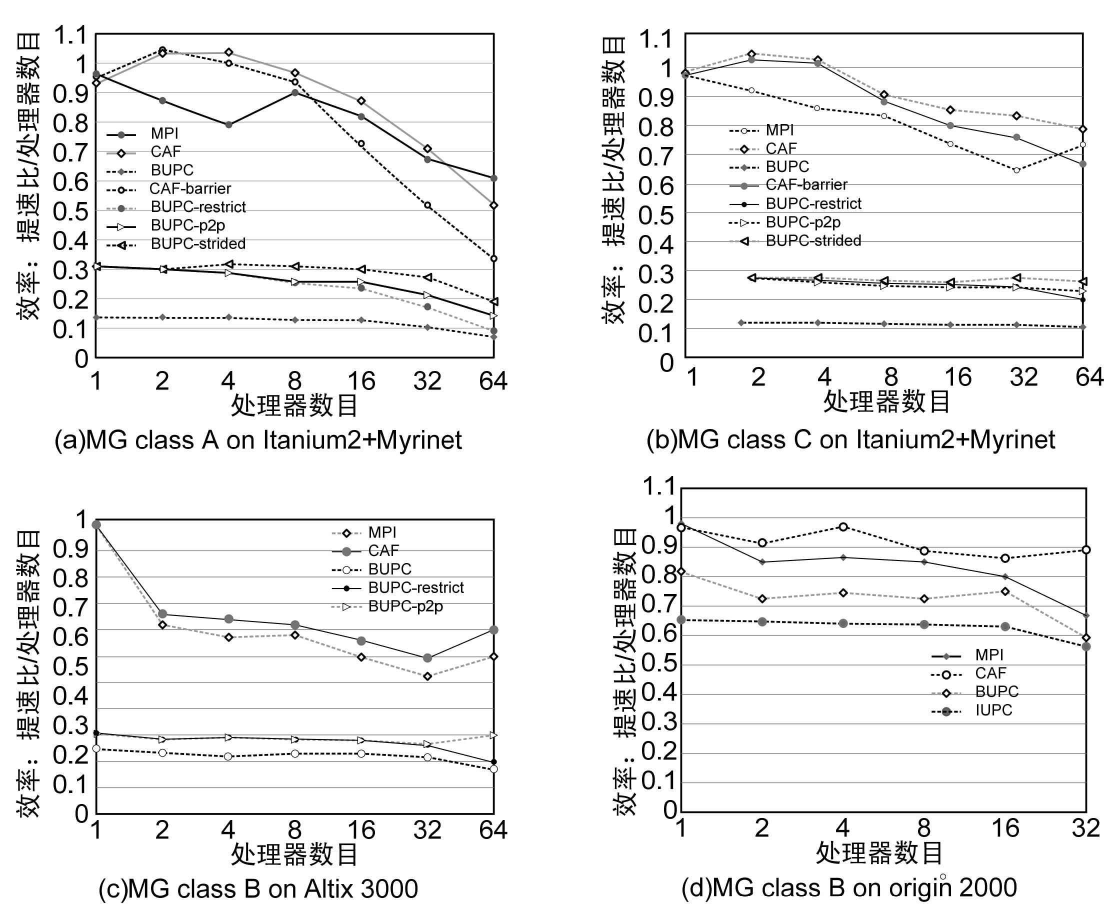
这里有一个问题，如代码的第 4 行，每个镜像都申请了一个名为 `work` 的局部数组来保存收集来的局部最小值。事实上，只有第一个镜像才用得到 `work` 数组。这里我们可以使用 CAF 提供的 **allocatable array** 来实现只在 `image 1` 中申请 `work` 数组。图 9 给出改进版的代码。如图 9 中的灰色填充框所示，先声明 `work` 数组为 **allocatable**，然后在 `image 1` 要执行的代码内部为其分配并释放空间。

2.8.2 Co-array 的实现情况

很久以来在许多 **Fortran** 编译器以及相应的系统中就实现了 Co-array 扩展，例如 Cray **Fortran**[27]自从 3.1 版本以来就包含了 Co-array 的支持。经过不断的发展和完善，Co-array 扩展已经被纳入了 **Fortran 2008** 的标准之中，且在越来越多的系统中得到实现和支持。第一个开源的 Co-array 实现，是在 G95 中把 Co-array 作为 **Fortran 2008** 的标准在编译器和相关的 Linux 运行时系统中实现。

此外，一些研究机构和公司也在积极地实现 CAF2.0 标准的编译器以及运行时库。例如莱斯大学截至到 2011 仍在开发支持 CAF2.0 的产品级的开源编译器，其 CAF2.0 的运行时库采用加州大学伯克利分校（UC Berkeley）的 GASNet[33]作为基础通讯库。GASNet 是一个和语言无关的，低层的网络通讯层，用来提供独立于网络的，高效的通讯原语，可以用来支持一些常见的全局地址空间的 SPMD 语言，例如 UPC、Titanium、**Co-Array Fortran** 等。

2.8.3 Co-array 的应用效果



(a)MG class A on Itanium2+Myrinet
(b)MG class C on Itanium2+Myrinet
(c)MG class B on Altix 3000
(d)MG class B on origin 2000

图10. MPI, CAF 和 UPC 在 NAS MG 上的性能比较图

克里斯坦·克拉法（Cristian Coarfă）等人[34]于 2005 对于共享地址空间的 SPMD 语言进行了评估，其中包含 CAF。实验使用 NAS Parallel Benchmarks (NPB) MG, CG, SP 和 BT。比较了 CAF，UPC 和 **Fortran** + MPI 三种实现的效果。实验平台有 4 种：

1. 92 节点 HP zx6000 的工作站 + Myrinet 2000
2. Lemieux Alpha cluster， Pittsburgh Supercomputing Center，
3. SGI Altix 3000
4. SGI Origin 2000

图 10 显示了 MPI、CAF 和 UPC 在 NAS MG 上的性能比较，对比了 MPI **Fortran** 版本、CAF、UPC 以及其使用不同变种版本时的性能，其中 BUPC 代表使用 BUPC 编译，CAF-barrier 指使用闩锁（barrier）方式进行同步。BUPC-restrict 指使用 **restrict** 关键字以便别名优化，BUPC-p2p 指使用点对点（point-to-point）方式进行同步，BUPC-strided 使用 UPC 扩展对跨块数据（strided data）进行成批传输（bulk transfers）。

图 11 显示了 MPI、CAF 和 UPC 在 NAS CG 上的性能比较：

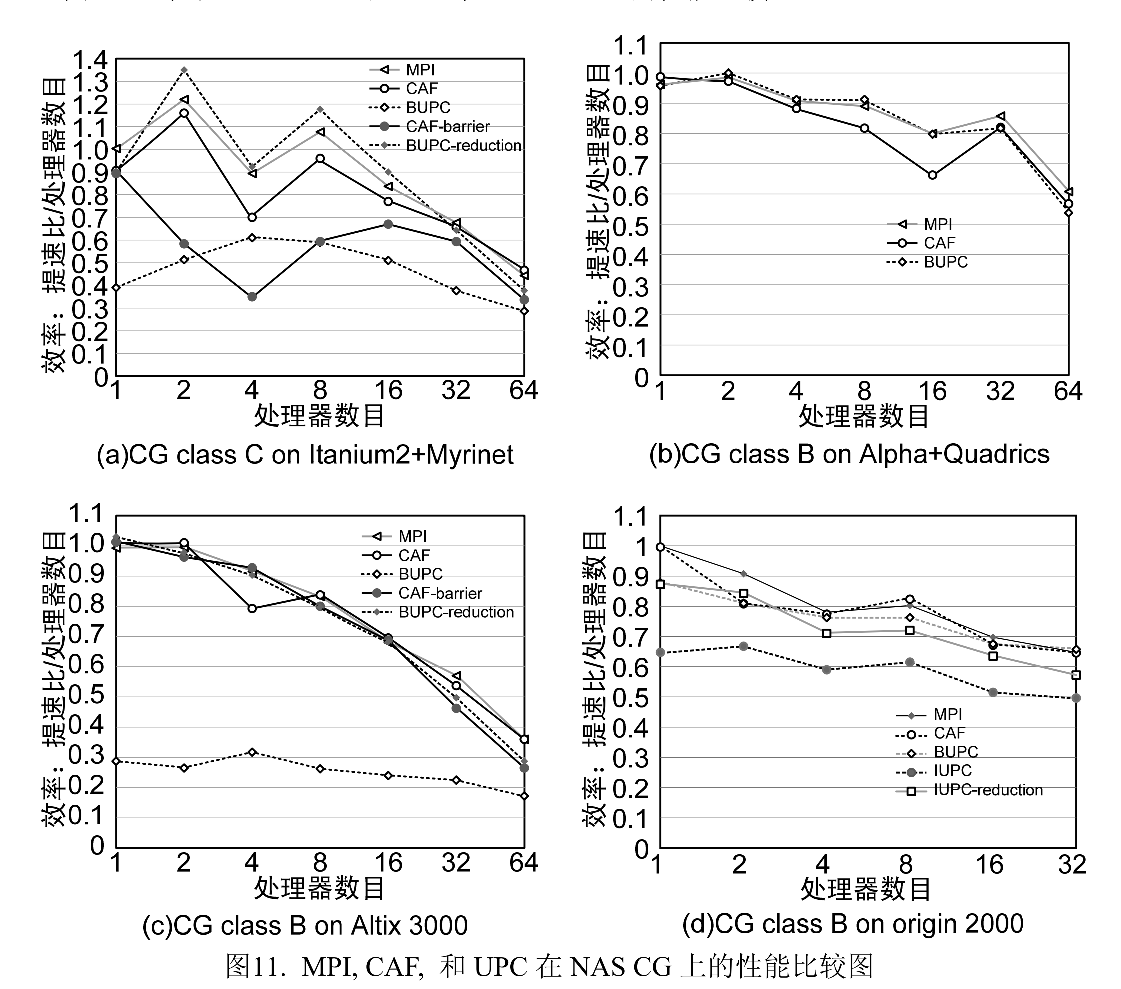


图11. MPI, CAF, 和 UPC 在 NAS CG 上的性能比较图

由图 10 和图 11，可以看出虽然性能比较的结果由于程序的差别而有所不同，但是 CAF 的性能和传统的 MPI 程序以及新兴的 UPC 语言都显示出了不错的效果。作为一门共享地址空间的并行语言扩展，CAF 使用最容易接受的，容易编写的方式对 **Fortran** 语言的数据对象下标进行扩展，且达到了很好的性能效果，并且有较好的扩展性和平台适应性，这也是 CAF 可以被加入 **Fortran 2008** 标准的一个重要原因。

2.9 其他

2.9.1 F--

作为 **Co-Array Fortran** 的前身，F--[35]是基于 **Fortran 77/90** 语言的并行扩展，同样使用 SPMD 编程模型，且其执行模型（计算分布）和数据模型（数据分布）和 Co-Array 完全相同，只是语法略有不同，部分原因是由于 **Fortran 90** 和 **95** 语法的区别。

2.9.2 Fortran D

Fortran D[13][36,37]是一组对 **Fortran 77** 语言的高效的并行扩展，可以使用在分布式内存的机器上指定数据划分，然后 **Fortran D** 编译器自动创建高效的并行代码，基于数据划分进行计算划分。编译器会产生显式的通信（以消息通信为基础），并且优化这些通信来生成高效的并行代码，最终数据会按照指定的数据划分分布在各个节点上，SPMD 的可执行程序则会分布到所有节点上并行执行。

Fortran D 尽管是一个很老的语言扩展和编译系统，且现在也逐渐不再使用了，但是它的发展对很多在分布式内存并行机上进行的并行程序研究做出了巨大的贡献。例如 **Fortran D** 对 HPF 的发展起过很重要的影响，HPF 采用了几乎同样的方式把数据划分的任务交给程序员来做，而且 HPF 的后端也采用很多相同的优化技术。这也是 HPF 和 **Fortran D** 扩展如此相似的原因。

2.9.3 MPF

MPF[38]编程语言是由阿•雅•卡里诺夫（A.Ya. Kalinov）等提出的对 **Fortran 90** 的一个扩展，是为分布式并行系统而设计的。MPF 的设计经验来自于 MPC 编程语言的发展和应用。MPF 是一个显式并行编程方法，试图寻找效率和易编程性之间的权衡。

在以往 **Fortran** 语言的并行扩展中，HPF 容易编程，程序员的开发效率较高，但是程序的编译优化较为困难，导致其性能很大程度依赖编译器的实现，并且比 MPI + **Fortran** 编写的程序的效率低。所以 MPF 的目的就是提供较好的编程性，且在效率上较 HPF 有所提高。

MPF 的特性不再详细介绍，和 MPI 程序在少量程序上的对比结果，MPI 程序的性能大概比 MPF 程序高 10%左右。而 MPF 程序的表达能力较好，编程容易。但 MPF 并没有广泛使用和被接受，仅仅处在研究实验阶段。

2.9.4 IPFortran

IPFortran[39]是一门利用多进程实现数据并行的编程语言，是 **Fortran** 语言的一个扩展。

实现多进程间数据并行通信时，`send`（发送）和 `receive`（接收）被隐式地包含在中缀运算符（插入运算符）以及 `reduce` 函数中。由于和系统相关的消息通信代码对程序员透明，由编译器和运行时来接管，代码串行编写，所以代码的开发效率提高。而且消息通信的逻辑是由编译器来自动生成，从而减少了开发中的错误。

IPFortran 编程有一个重要的前提是，每个处理器知道所有处理器上的变量的名字。为了达到这个前提，我们需要所有处理器运行相同的 **IPFortran** 代码，编程模型是 SPMD。程序员使用数据的局部视图（local view）完成数据的分发和读取。

程序员通过局部化（local）方法使用显式的逻辑为每个处理器写代码，且使用相应的控制结构来进行数据的分解和传播。使用这种方法程序员可以高效地写出 **IPFortran** 代码，且

很多工业级的代码都可以很快地移植到 **IPFortran** 上来。

2.9.5 Fortran M

Fortran M[40,41]是在 **Fortran 77** 的基础上进行的一组扩展，它支持面向模块化的消息传递的程序设计方法。**Fortran M** 具有以下特点：

- **模块化** 程序使用显式定义的通信通道进行通信。**Fortran M** 程序就是由这些通信通道将模块连接在一起而形成的。每个模块称为一个进程（process），每个进程可以包含普通数据、子进程，以及内部通信。
- **安全** 在通道上的操作经过严格的限制和规定，可以避免二义性的执行和不正确的结果。通信的通道是有类型的，所以编译器可以检查其使用是否正确。
- **不依赖于体系结构** 进程映射时，可以先映射到一个和实际机器不同的虚拟计算机配置上。映射可以通过注释进行，结果只会影响程序的性能，不会影响正确性。
- **高效** **Fortran M** 可以在单处理器、共享内存的系统、非共享内存分布式的系统以及网络工作站上高效地实现。由于通信被包含在代码之中，编译器也可以在优化计算时优化通信。

2.9.6 OpenMP Fortran

OpenMP[42]并非专属 **Fortran** 的扩展，而是一套工业标准的制导信息、库函数和环境变量，其目标在于提供一套得到编译器广泛支持的共享内存机器上的并行程序开发方法。与 HPF 类似，OpenMP 通过制导信息指定并行域，并行域中的代码将在线程上执行。OpenMP 的并行域允许嵌套。

OpenMP 得到了几乎所有主流编译器的支持，已经成为共享内存环境中并行编程的事实标准，并因其突出的可移植性而逐渐取代了很多厂商的私有扩展。

3 厂家私有的并行扩展

3.1 VPP Fortran

VPP Fortran[43~46]最初是为 VPP 系列超级计算机开发的（1991）。一方面，由于使用全局名字空间，**VPP Fortran** 编程比 MPI 容易得多，数据可以分解和分布在多个处理器上，通过运行时来保证数据的一致性和同步等问题。另一方面，**VPP Fortran** 提供较为低层的方法来支持纯数据并行，从而得到更好的性能。这些方法可以显式地存取每个处理器上的数据，实现广播、多播、栅栏、关键区、同步等等操作。这些低层的特性不是任何时候都是必要的，但是他们可以用来提升关键程序的性能。

3.1.1 数据分布

VPP Fortran 编程模型描述的目标硬件如图 12 所示：每个处理器有自己的本地存储器，本地存储器总是最快的存储器。全局存储器物理上分布在每个处理器上，被所有的处理器共享使用。在 **VPP Fortran** 编程模型中，每个变量要么是局部（Local），要么是全局（Global）属性，取决于其所在的位置。

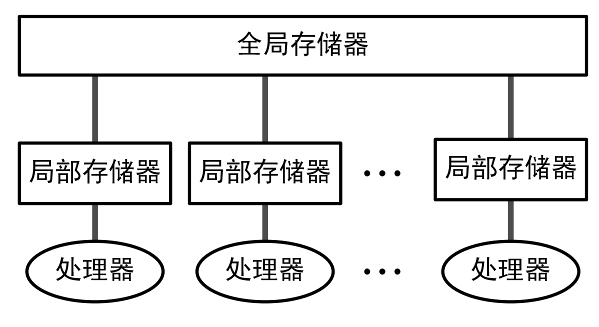


图12. **VPP Fortran** 编程模型

3.1.2 执行过程

以下使用如图 13 所示的一段代码来说明 **VPP Fortran** 程序的执行过程：

包含在 **PARALLEL REGION**（并行区域）之中的代码段将会被并行执行。**PARALLEL REGION** 将会在每个处理器上使用进程分叉函数（`fork`） 来创建进程，并在 **END PARALLEL** 的时候调用 `join`，这种行为和 OpenMP 的行为有些类似。但是 **VPP Fortran** 是使用在分布式内存的并行系统上，控制和数据都是要分布在每个节点处理器上；而 OpenMP 则是使用在共享内存的系统上，只有控制是分布在多个处理器上的。

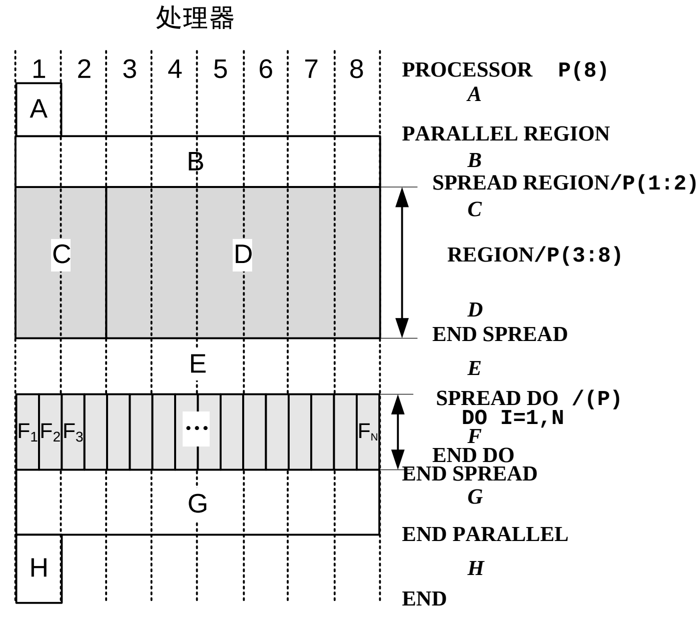


图13. VPP **Fortran** 执行的例子

VPP Fortran 有两种分解（SPREAD）构造，一种是 **SPREAD REGION**， 是用来描述任务如何分解的；一种是 **SPREAD DO**，是用来描述循环（LOOP）是如何分解的。一组 **PARALLEL REGION** 构造或者 **SPREAD** 构造和一组处理器相对应，称为区域（Region)，区域可以嵌套。如图 13：C 被分派到了处理器 1 和 2 上，D 被分派到了处理器 3-8 上。**SPREAD DO** 则把循环的各个迭代分派到所有的处理器上并行执行。此外 VPP **Fortran** 还可以实现分区索引（index partition）和显式通信。

3.2 HPF/JA extension

HPF/JA extensions[47~50]是基于 **VPP Fortran** 的特性，在 HPF2.0 的基础上所做的系列扩展。其特性包括处理期间的异步通信、显式的编译制导 shadow、local directive（局部目录）等。这些特性也是 **VPP Fortran** 中在处理真实应用时非常有用的。这里给出一个 HPF/JA 的例子来讲解一下其特性：

图 14 展示了一个如何在相邻处理器上存取数据的例子，其中左边是 BT 基准程序的 HPF 实现。X 和 U 数组的第 2 维按照块（BLOCK）的方式进行分布，并且对下面的循环进行并行。访问数组 X 是本地访问，但是 U 数组就不一定了，这是由于 U 数组的下标访问为 J-1、J-2、J+1、J+2 而不是 J，而这些元素恰好会被映射到相邻的处理器上。HPF2.0 通过定义 shadow 编译制导来用于这样类似的循环，shadow 指定了该处理器在其本地的存储器上有一块内存可以保存相邻处理器的一部分数据，以方便存取。

图 14(b)显示了如何利用 shadow 编译制导，其中 U 的第 2 维上下边界都有 2 个元素大小的 shadow 区域。Reflect 制导在循环前被使用用来将相邻处理器的边界元素拷贝到当前处理器的本地存储上，这就保证 U 被存取之前是局部的，使我们可以用 local 编译制导。

其中 local 编译制导被引入的原因如下：即使当执行的时候数据是存放在同一个处理器上，而编译器在编译的时候不知道数据在哪里，还是有可能造成存取低效。原因是在运行时

找到数据的位置将会涉及到大量的指令执行，开销比较大。尽管数据存放的位置大多数时候可以在编译的时候被分析出来，但是仍旧会有一些不确定性，local 编译制导被引入是用来告诉编译器数据确定是存放在当前处理器的本地存储上，从而优化数据存取，提高执行效率。

```
!hpf$ processors p(4)
!hpf$ distribute (*,block)
!hpf$& onto p :: x,u
                        :
!hpf$ independent,new(i,j)
P  do 100 j=3,n-2
P  !hpf$ on home(x(:,j)) begin
P   do 200 i=1,n
P 5 x(i,j)=x(i,j)+u(i,j-2)
P &         +u(i,j-1)+u(i,j)
P &         +u(i,j+1)+u(i,j+2)
P 200 end do
P  !hpf$ end on
P 100 end do
```

（a） 朴素编码

```
    !hpf$ processors p(4)
    !hpf$ distribute (*,block)
    !hpf$&  onto p:: x,u
    !hpf$ shadow u(0,2:2)
                            :
    !hpfj reflect u
    !hpf$ independent,new(i,j)
    P   do 100 j=3,n-2
    P  !hpf$ on home(x(:,j)) begin
    P  !hpfj local begin
V  P  do 200 i = 1,n
V  P   x(i,j)=x(i,j)+u(i,j-2)
V  P   &        +u(i,j-1)+u(i,j)
V  P   &        +u(i,j+1)+u(i,j+2)
V  P   200  end do
    P  !hpfj end local
    P  !hpf$ end on
    P  100  end do
```

（a） 利用 shadow 编译

图14. HPF/JA 和 HPF2.0 对比的例子（在相邻处理器上存取数据）

3.3 ICL DAP Fortran

International Computing Limited 公司生产了最早的并行处理器 DAP，并为之开发了 **DAP Fortran**。**DAP Fortran**[51]是针对 **FORTRAN 77** 非读写（I/O）部分的扩展，使之支持数组、矩阵计算。

3.4 PGI[6] CUDA Fortran

CUDA(Compute Unified Device Architecture) [52]是显卡厂商英伟达（nVIDIA）推出的运算平台。 CUDA 是一种通用并行计算架构。该架构包含了 CUDA 指令集架构（ISA）以及 GPU 内部的并行计算引擎，使 GPU 能够解决复杂的计算问题。开发人员现在可以使用 C 语言来为 CUDA 架构编写程序。所编写出的程序可以在支持 CUDA 的处理器上以超高性能运行。

图 15 和图 16 展示了一个分块矩阵乘法的例子：

CUDA 支持 **FORTRAN** 以及 C++。**CUDA Fortran** 和 CUDA C 类似，**CUDA FORTRAN** 是英伟达和 PGI 共同制定，在 PGI 的 **FORTRAN** 编译器中实现的。是基于编译制导的，类似 OpenMP 的接口。它仍然遵循简单的异构环境的编程接口。CPU 和 GPU 是不同的设备，拥有不同的地址空间，宿主（HOST）代码运行在 CPU 之上，设备（DEVICE）代码运行在

[6] Premiere Global Services，一家在全球范围内提供通信技术的虚拟协作和会议解决方案的跨国公司

GPU 上。

The Portland Group 公司和英伟达公司合作，定义了一个小的 **Fortran** 扩展集合，使得程序员可以在 **Fortran** 中直接对 CUDA 平台进行编程，通过扩展 **Fortran 90** 的定义变量属性的语法、内存分配语句和数组赋值来自然地引入对 CUDA 的支持。

```
subroutine mmul(A,B,C )
!
  use cuda for
  real, dimension(:,;)::A,B,C
  integer::N,M,L
  real, device, allocatable, dimension(:,;)::Adev,Bdev,Cdev
  type(dim3)::dimGrid,dimBlock
!
  N=size(A,1); M=size(A,2); L=size(B,2)
  allocate(Adev(N,M),Bdev(M,L),Cdev(N,L))
  Adev=A(1:N,1:M)
  Bdev=B(1:M,1:L)
  dimGrid=dim3(N/16,L/16,1)
  dimBlock=dim3(16,16,1)
  call mmul_kernel<<<dimGrid,dimBlock>>>(Adev,Bdev,Cdev,N,M,L)
  C(1:N,1:M) = Cdev
  deallocate(Adev, Bdev, Cdev )
!
  end subroutine
```

图15. **CUDA Fortran** 调用 GPU kernel 示例

```
  attributes(global) subroutine MMUL_KERNEL( A,B,C,N,M,L)
!
  real,device :: A(N,M),B(M,L),C(N,L)
  integer,value :: N,M,L
  integer :: i,j,kb,k,tx,ty
  real,shared :: Ab(16,16),Bb(16,16)
  real :: Cij
!
  tx=threadidx%x ; ty=threadidx%y
  i=(blockidx%x-1)*16+tx
  j=(blockidx%y-1)*16+ty
  Cij= 0.0
  do kb=1,M,16
     ! 从 Ab 和 Bb各读取一个元素; 注意在本进程块中有 16x16 = 256 个进程在分别读
       取Ab和Bb的元素
     Ab(tx,ty)=A(i,kb+ty-1)
     Bb(tx,ty)=B(kb+tx-1,j)
     ! 等候直到Ab和Bb的所有元素都被填充
```

```
    call syncthreads()
    do k=1,16
      Cij=Cij+Ab(tx,k)*Bb(k,ty)
    enddo
    ! 等候直到进程块内在本次迭代中涉及Ab和Bb的所有进程全部完成
        call syncthreads()
  enddo
  c(i,j) = Cij
!
  end subroutine
```

图16. CUDA **Fortran** GPU kernel 示例

3.5 Cray Parallelization Directives

克雷（Cray）公司的 **Cray Fortran**[53]编译工具链中包括了大量该公司的私有制导信息的支持。这些制导信息除了可以用来控制编译器特性，还可以用来标记需要自动线程并行的循环等。

3.6 Sun Parallelization Directives

Oracle 的 Sun Studio 开发环境[54]中除了对于标准的 OpenMP 有支持之外，亦有对于 Cray 和 Sun 私有格式的，用于 **Fortran 95** 之后的 FORALL 结构上的并行制导的少量支持。

4 总结

本文综述了自 **Fortran 77** 以来，对于 **Fortran** 语言的并行扩展，其中最主要的为 HPF 和 Co-array **Fortran**，应用广泛，支持较多。

Fortran 语言是一种极具发展潜力的语言，在全球范围内流行过程中，**Fortran** 语言的标准化不断吸收现代化编程语言的新特性，并且在工程计算领域仍然占有重要地位。

在数值计算中，**Fortran** 语言仍然不可替代。**Fortran 90** 标准引入了数组计算等非常利于矩阵运算的功能。在数组运算时，**Fortran** 能够自动进行并行运算，这是很多编程语言不具备的。**Fortran** 语言使用户能够运用很多现成的函数软件包，所以非常便利。

Fortran 语言并没有因为其历史悠久，跟不上发展的脚步而走向没落，反而逐渐引入了新语言的特性，加入新硬件的支持，改进旧的并行扩展等等，例如 Co-Array 在近几年得到广泛支持并成为 **Fortran 2008** 的标准。**Fortran** 对新兴加速器，例如 GPU CUDA 编程的支持等等说明其正与时俱进的发展着。且由于 **Fortran** 语言一直在高性能计算领域占据主要地位，各种工具和高性能计算软件包，数学库等比较完善，为其今后的发展提供了良好的基础。

参考文献：

[1] Backus, J. W., H. Stern, I. Ziller, et al. (1957). "The **FORTRAN** Automatic Coding System". Western joint computer conference: Techniques for reliability (Los Angeles, California: Institute of Radio Engineers, American Institute of Electrical Engineers, ACM): 188–198. doi:10.1145/1455567.1455599.

[2] 段朝晖. 1995 年. **FORTRAN** 语言的发展历史. *计算机世界报*, 第 8 期. 国家智能计算机研究开发中心.

[3] ANSI x3.9-1966. USA Standard **FORTRAN**. American National Standards Institute. Informally known as **FORTRAN 66**.

[4] ANSI x3.9-1978. American National Standard – Programming Language **FORTRAN**. American National Standards Institute. Also known as ISO 1539-1980, informally known as **FORTRAN 77**.

[5] ANSI X3.198-1992 (R1997) / ISO/IEC 1539:1991. American National Standard – Programming Language **Fortran** Extended. American National Standards Institute / ISO/IEC. Informally known as **Fortran 90**.

[6] ISO/IEC 1539-1:1997. Information technology – Programming languages – **Fortran** – Part 1: Base language. Informally known as **Fortran 95**. There are a further two parts to this standard. Part 1 has been formally adopted by ANSI.

[7] ISO/IEC 1539-1:2004. Information technology – Programming languages – **Fortran** – Part 1: Base language. Informally known as **Fortran 2003**.

[8] ISO/IEC 1539-1:2010 (Final Draft International Standard). Information technology – Programming languages – **Fortran** – Part 1: Base language. Informally known as **Fortran 2008**.

[9] Page, Clive G. (1988). Professional Programmer's Guide to **Fortran77** (7 June 2005 ed.). London: Pitman. ISBN 0-273-02856-1. Retrieved 4 May 2010.

[10] CORPORATE The Parallel Computing Forum (1991). ACM SIGPLAN **Fortran** Forum. Volume 10 Issue 3, Sept. 1991, Pages 1 – 57, ACM New York, NY, USA.

[11] Press, William H. (1996). Numerical Recipes in **Fortran 90**: The Art of Parallel Scientific Computing. Cambridge, UK: Cambridge University Press. ISBN 0-521-57439-0.

[12] Akin, Ed (2003). Object Oriented Programming via **Fortran 90/95** (1st ed.). Cambridge University Press. ISBN 0-521-52408-3.

[13] D. Tam, Prof. Abdelrahman, (2000). **Fortran D** - Final Report. Compilation Techniques for Parallel Processors ECE 1754. Friday May 12, 2000.

[14] M. Ujaldon, E. L. Zapata, B. M. Chapman, et al. (1997). **Vienna-Fortran**/HPF Extensions for Sparse and Irregular Problems and Their Compilation. IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, vol. 8, pages: 1068 – 1083, 1997.

[15] HPFF - Rice University HPF Forum, http://hpff.rice.edu/ (http://dacnet.rice.edu/).

[16] **High Performance Fortran** Forum (1994). **High Performance Fortran** Language Specification.

[17] C. H. Koelbel. (1997). The high performance **Fortran** handbook – 2nd edition. MIT press.

[18] Shires Dale, Mohan Ram. An Evaluation of HPF and MPI Approaches and Performance in Unstructured Finite Element Simulations. Journal of Mathematical Modeling and Algorithms 2002-09-01.

[19] Murai, Hitoshi, Araki, et al. Implementation and evaluation of HPF/SX V2. Concurrency and Computation: Practice and experience.

[20] Y. C. Hu, G. H. Jin, S. L. Johnsson, et al. HPFBench: A **High Performance Fortran** Benchmark Suite.

[21] Applied Parallel Research, Inc., APR's public domain benchmark suite, ftp://ftp.infomall.org/tenants/apri/Benchmarks, 1996.

[22] C. A. Addison, et al., The GENESIS Distributed-memory Benchmarks, Computer Benchmarks, J.J. Dongara and W. Gentzsch (Eds), Advances in Parallel Computing, Vol. 8, Elsevier Science Publications, BV (North Hol-land), Amsterdam, The Netherlands, pp. 257-271, 1991.

[23] D. E. Bailey, et al., The NAS Parallel Bench-marks, Technical Report RNR-94-007, NASA Ames Research Center, 1994.

[24] Chapman, Stephen J. (2007). **Fortran 95/2003** for Scientists and Engineers (3rd ed.). McGraw-Hill. ISBN 978-0-07-319157-7.

[25] Chivers, Ian; Sleightholme, Jane (2012). Introduction to Programming with **Fortran** (2nd ed.). Springer. ISBN 978-0-85729-232-2.

[26] Metcalf, Michael; John Reid, Malcolm Cohen (2011). Modern **Fortran** Explained. Oxford University Press. ISBN 0-19-960142-9.

[27] **Co-array Fortran**: http://www.co-array.org/.

[28] **Co-array Fortran** at Rice: http://caf.rice.edu/.

[29] R. W. Numrich, J. Reid. (1998). **Co-array Fortran** for parallel programming. SIGPLAN **Fortran** Forum, 17(2): 1-31, Aug. 1998.

[30] R. W. Numrich, J. Reid, (2005). Coarrays in the next **Fortran** Standard. SIGPLAN **Fortran** Forum, 24(2): 4-17, Aug. 2005.

[31] C. Coarfa, Y. Dotsenko, J. Eckhardt, et al. **Co-array Fortran** performance and potential: An NPB experimental study. In 16th International Workshop on Languages and Compilers for Parallel Processing (LCPC), October 2003.

[32] John Mellor-Crummey, Laksono Adhianto, William Scherer III, and Guohua Jin, A New Vision for **Co-array Fortran**, Proceedings PGAS09, 2009.

[33] GASNet: http://gasnet.cs.berkeley.edu/.

[34] C. Coarfa, Y. Dotsenko, J. Mellor-Crummey, et al. 2005. An evaluation of global address space languages: **co-array Fortran** and unified parallel C. In Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming Pages 36 – 47, ACM New York, NY, USA (PPoPP '05).

[35] R. W. Numrich, J. L. Steidel, B. H. Johnson,et al. (1997). Definition of the F-- extension to **Fortran 90**. In Proceedings of the 10th International Workshop on Languages and Compilers for Parallel Computing (LCPC '97), Pages 292 – 306, Springer-Verlag London, UK.

[36] S. Hiranandani, K. Kenney, C. Koelbel, et al. (1991) An Overview of the **Fortran D** Programming System. In Proceedings of the Fourth International Workshop on Languages and Compilers for Parallel Computing, Pages 18 – 34, Springer-Verlag London, UK, 1991.

[37] R. V. Hanxleden, K. Kennedy, C. Koelbel, et al. (1992). Compiler Analysis for Irregular Problems in **Fortran D**. In Proceedings of the 5th International Workshop on Languages and Compilers for Parallel Computing, Pages 97 – 111, Springer-Verlag London, UK, 1992.

[38] Kalinov, Alexey, Ledovskih, et al. (2004). An Extension of **Fortran** for High Performance Parallel Computing. Programming and Computer Software. 2004.

[39] B. Bagheri, T. W. Clark, L. R. Scott. 1992. **IPFortran**: a parallel dialect of **Fortran**. SIGPLAN **Fortran** Forum 11, 3 (September 1992), 20-31. DOI=10.1145/141438.141450.

[40] I. T. Foster, K. M. Chandy. (1995). **Fortran M**: A Language for Modular Parallel Programming. Journal of Parallel and Distributed Computing, Volume 26, Issue 1, 1 April 1995, Pages 24–35.

[41] I. T. Foster, D. W. Walker. (1994). Paradigms and Strategies for scientific computing on distributed memory concurrent computers. In Proceedings of the High Performance Computing 1994 Conference, La Jolla, CA, April 11-15, 1994. Published by the Society for Computer Simulation, San Diego, CA.

[42] OpenMP: http://openmp.org/wp/.

[43] H. Iwashita, S. Okada, M. Nakanishi, et al. (1994). **VPP Fortran** and Parallel Programming on the VPP500 Supercomputer. In Proceedings of the 1994 International Symposium on Parallel Architectures, Algorithms and Networks (poster session papers), pages 165-172. Japan, December 1994.

[44] E. Yamanaka, T. Shindo. (1997). Parallel Language Processing System for High-Performance Computing. Fujitsu Sci. Tech. J., 33(1): 39-51, June 1997.

[45] H. Iwashita. (1997). A view of **VPP Fortran** in contrast with HPF. IPSJ Magazine, 38(2), February 1997. (in Japanese)

[46] **Fortran** VPP300 User guide: Programming in **VPP Fortran**. http://anusf.anu.edu.au/VPP/Userguide/VPP**Fortran**.html.

[47] HPF/JA Language Speciation, JAHPF (Japan Association for **High Performance Fortran**), January 31, 1999 Version 1.0.

[48] Y. Seo1, H. Ohta, H. Sakagami, et al. (2002). HPF/JA: extensions of **High Performance Fortran** for accelerating real-world applications Concurrency and Computation: Practice and Experience Volume 14, Issue 8-9, pages 555–573, 2002.

[49] H. Iwashita1, N. Sueyasu1, S. Kamiya1, et al. (2002). **VPP Fortran** and the design of HPF/JA extensions. Concurrency and Computation: Practice and Experience Special Issue: **High Performance Fortran** Volume 14, Issue 8-9, pages 575–588, 2002

[50] Asaoka, Kae, Hirano, et al. Evaluation of the HPF/JA Extensions on Fujitsu VPP Using the NAS Parallel Benchmarks High Performance Computing.

[51] ICL. Icl dap **Fortran**: http://www.hpjava.org/talks/beijing/hpf/introduction/node5.html.

[52] CUDA: http://developer.nvidia.com/cuda/nvidia-gpu-computing-documentation.

[53] **Cray Fortran**: www.ccs.ornl.gov/Phoenix/ftn.html.

[54] Oracle Sun Studio: www.oracle.com/.

作者简介：

苏　乐： 中科院计算所，前瞻技术实验室，虚拟现实技术课题组，博士在读，sule@ict.ac.cn
房双德： 中科院计算所，计算机体系结构国家重点实验室，先进编译技术组，博士在读
黄元杰： 中科院计算所，计算机体系结构国家重点实验室，先进编译技术组，博士在读